

10/552733



| | |
|-------------------|-----|
| REC'D 08 DEC 2003 | |
| WIPO | PCT |

**Prioritätsbescheinigung über die Einreichung
einer Patentanmeldung**

Aktenzeichen:

102 50 639.6

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH
RULE 17.1(a) OR (b)

Anmeldetag:

30. Oktober 2002

Anmelder/Inhaber:

Siemens Aktiengesellschaft, München/DE

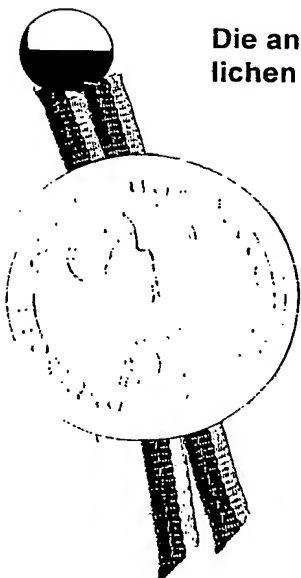
Bezeichnung:

Verwaltung von mit einer erweiterbaren Auszeich-
nungssprache beschriebenen Daten

IPC:

G 06 F 17/30

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprüng-
lichen Unterlagen dieser Patentanmeldung.



München, den 30. Oktober 2003
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Schäfer

Beschreibung

Verwaltung von mit einer erweiterbaren Auszeichnungssprache beschriebenen Daten

5

Die Erfindung betrifft ein Verfahren sowie ein System zur Verwaltung von mit einer erweiterbaren Auszeichnungssprache beschriebenen Daten.

- 10 Daten werden häufig mit einer erweiterbaren Auszeichnungs-
sprache beschrieben. Eine solche Auszeichnungssprache ist
z. B. XML (= Extensible Markup Language). Dieses textbasierte
Format wird sowohl als Austauschformat als auch als
15 Speicherformat verwendet. Ein Nachteil des Formats ergibt
sich dadurch, dass das Datenvolumen durch dieses Ablageformat
sehr schnell sehr umfangreich werden kann. In der Datenablage
werden oft Objekte abgelegt (z. B. Objekte aus der
Automatisierungswelt). Sollen diese wieder eingelesen werden,
so kann dies recht aufwendig sein. Insbesondere wenn sich
20 eine Applikation nur für eine Teilmenge der Objekte bzw. nur
einen Teil der Daten interessiert. Es muss dennoch immer die
gesamte Datei sequenziell gelesen und bearbeitet werden, da
mit erweiterbaren Auszeichnungssprachen Daten in Dateien
stream-orientiert abgelegt und verarbeitet werden.

25

Der Erfindung liegt die Aufgabe zugrunde, die Verwaltung von
mit einer erweiterbaren Auszeichnungssprache beschriebenen
Daten zu vereinfachen.

- 30 Diese Aufgabe wird durch ein Verfahren zur Verwaltung von mit
einer erweiterbaren Auszeichnungssprache beschriebenen Daten
gelöst, wobei die Daten in Form von Objekten strukturiert
werden, wobei Bestandteile der Objekte in ersten Dateien
speicherbar sind, wobei die Bestandteile jeweils eine
35 logische Einheit eines Objekts abbilden und wobei eine zweite
Datei mit ersten Mitteln zur Referenzierung der Bestandteile

als übergeordnete, objektbasierte logische Ebene zur Speicherung der Objekte vorgesehen ist.

5 Diese Aufgabe wird durch ein System zur Verwaltung von mit
einer erweiterbaren Auszeichnungssprache beschriebenen Daten
gelöst, wobei Objekte zur Strukturierung der Daten vorgesehen
sind, wobei Bestandteile der Objekte in ersten Dateien
speicherbar sind, wobei die Bestandteile jeweils eine
logische Einheit eines Objekts abbilden und wobei eine zweite
10 Datei mit ersten Mitteln zur Referenzierung der Bestandteile
als übergeordnete, objektbasierte logische Ebene zur
Speicherung der Objekte vorgesehen ist.

15 Objektgeflechte werden oft in einer großen Datei abgelegt
oder aber auf eine Vielzahl kleiner Dateien aufgeteilt.
Zusammenhänge zwischen Objekten sind entweder durch die
Ablagestruktur vorgegeben oder aber durch Links, die Verweise
zwischen den Dateien und darin befindliche Objekte
repräsentieren, abgebildet. Die Erfindung schlägt ein
20 Verfahren sowie ein System vor, mit dem man die Ablage von
Objekten und Objektgeflechten auf mehrere Dateien aufteilen
kann. Der Zugriff auf das Objektgeflecht wird dabei
optimiert. Die Anzahl zu lesender Dateien - und damit die
Datenmenge, die gelesen werden muss - wird reduziert.

25 Grundlage hierfür ist, dass über der Ebene mit in einer
reinen Auszeichnungssprache beschriebenen Daten eine weitere,
logische Ebene für Objekte definiert wird. D. h. ein
Verfahren bzw. System zur Abbildung von Objekten mit ihren
Daten in der Auszeichnungssprache. Applikationen, die die
30 Daten lesen, müssen nicht das gesamte Objektgeflecht und
dessen Daten lesen, sondern können die logische Objektebene
nutzen um nur bis bis zu der Granularität lesen, die sie für
ihre Arbeiten auf dem Objektgeflecht benötigen. Tools, die
bestimmte Teile des Objektgeflechts nicht benötigen, können
35 so die entsprechenden Stellen sehr leicht überlesen, da die
Daten bzw. die Informationen in separaten Auslagerungsdateien

abgelegt werden. Diese Teile müssen vom Parser der Auszeichnungssprache nicht eingelesen und bearbeitet werden. Es können Teile (im Folgenden auch Features genannt) eines Objekts in erste Dateien ausgelagert werden. In der
5 jeweiligen ersten Datei werden ein oder mehrere ausgelagerte Objektteile abgelegt. In der Ursprungsdatei verbleibt in diesem Fall das Objekt. Nur ein oder mehrere Features des Objekts werden ausgelagert. Dies ermöglicht Navigierbarkeit auf das Objekt innerhalb der Ursprungsdatei bis hin zum
10 ausgelagerten Objektteil (Feature). Zudem bleiben die Objektteile verschiebbar, ohne dass Referenzen hierauf geändert werden müssen.

Gemäß einer vorteilhaften Ausgestaltung der Erfindung sind
15 die ausgelagerten Bestandteile selbst Objekte. In der zweiten Datei, auch Ursprungsdatei genannt, verbleibt jeweils nur ein Objektrumpf in Form einer Auslagerungsreferenz. Dies stellt sicher, dass Referenzen auf das ausgelagerte Objekt sich nicht von anderen Objektreferenzen, die Objekte oder
20 Objektteile in der Ursprungsdatei referenzieren, unterscheiden. Es muss am Ursprung der Referenz nicht bekannt sein, dass es sich bei dem Zielobjekt um ein ausgelagertes Objekt handelt. In der jeweiligen ersten Datei, auch Auslagerungsdatei genannt, wird das ausgelagerte Objekt als
5 Ganzes abgelegt. Ein Objekt kann somit verschoben werden, ohne dass Referenzen auf das Objekt zu ändern sind. Des Weiteren wird Navigierbarkeit auf das Objekt innerhalb der Ursprungsdatei bzw. von Außen ermöglicht.

30 Vorteilhafterweise werden die Features genannten Bestandteile der Objekte in objektspezifischen generischen Containern gespeichert, wobei die Container zur Referenzierung des jeweiligen Objekts dienen. In der Auslagerungsdatei werden die Features also in einem Container, im Folgenden auch
35 Stellvertreterobjekt oder ObjektSurrogate genannt, abgelegt. Dieses Stellvertreterobjekt repräsentiert in generischer Weise eine Hülle für die Daten des Objekts und bildet den

Kontext für die Ablage der Features. Der Kontext ist dabei die Identifikation des Objekts, wie es in der Ursprungsdatei identifiziert wird. Es gibt somit ein Stellvertreterobjekt in der Auslagerungsdatei, das beschreibt zu welchem Objekt in der Ursprungsdatei die Daten gehören. Das Stellvertreterobjekt ist ein Objekttyp, der ein generisches Objekt repräsentiert und beliebige Features aufnehmen kann. In der Auslagerungsdatei stehen die Daten des Objekts nicht alleine, sondern haben durch das Stellvertreterobjekt einen Bezug zum eigentlichen Objekt in der Ursprungsdatei und stellen somit eine Rückwärtsreferenz zur Verfügung.

Nachfolgend wird die Erfindung anhand der in den Figuren dargestellten Ausführungsbeispiele näher beschrieben und erläutert.

Es zeigen:

FIG 1 eine schematische Darstellung der Auslagerung eines Objekts in eine Auslagerungsdatei und

FIG 2 eine schematische Darstellung der Auslagerung eines Teils eines Objekts in eine Auslagerungsdatei.

In den Ausführungsbeispielen wird XML als Beispiel für eine erweiterbare Zeichnungssprache verwendet. Daten in XML-Dateien werden sequenziell gelesen und nicht benötigte Teile der Datei überlesen. Dabei ist die XML-Syntax sehr hilfreich. Sie sieht vor, dass Daten immer mit einem Start- und Ende-Tag gleichen Namens versehen werden (z. B. <DisplayName>) bzw. das Tag sofort wieder geschlossen wird (z. B. <Text .../>)

Beispiel:

```
<DisplayName>  
  <Text Value="DP-Master" />  
</DisplayName>
```

Somit ist es für das einlesende Tool („Parser“ genannt) möglich, Daten beginnend bei einem bestimmten Start-Tag bis zum zugehörigen Ende-Tag zu überlesen. Der Inhalt des Files zwischen den Tags muss jedoch dennoch gelesen werden, auch wenn die Daten nicht verarbeitet werden. Eine Methode zum Aufteilen von Datenbeständen auf mehrere File bietet das vom W3C Konsortium vorgeschlagene XML-Inclusions (XInclude) Konstrukt. Dies gehört zu den Basisdefinitionen von XML, die vom W3C (= World Wide Web Consortium) momentan erarbeitet werden. XInclude funktioniert als einfacher Mechanismus, um XML oder Textdateien in ein XML Dokument zu inkludieren. Dies geschieht analog dem aus C/C++ bekannten #include als textuelle Ersetzung des Xinclude-Tags durch das andere Dokument. Es können dabei entweder das gesamte Dokument oder nur Teile daraus (spezifiziert durch einen XPointer, siehe XML-Spezifikation) eingebettet werden. Dies löst jedoch nicht das Problem des Überlesens von nicht benötigten Objektteilen, da XML-Parser automatisch die referenzierten Dateien beim Lesen mit einfügen. Die zu hantierende Datenmenge bleibt dabei gleich. Beim Überlesen von nicht interessierenden Teilen der Datei gilt dasselbe wie oben beschrieben. Das Problem hierbei ist, dass XML an sich nur Daten repräsentiert und kein Objektmodell kennt. Somit können logisch in Objekten zusammenhängende Daten auf XML Ebene nicht erkannt werden. Eine weitere, heute gebräuchliche Möglichkeit ist die Aufteilung großer Datenfiles auf eine Vielzahl kleiner Dateien. Hierbei wird typischerweise so vorgegangen, dass die Grenze zwischen Dateien auch immer die logische Objektgrenze bildet. Damit werden Objekte der Anwendungsebene in einer Datei abgelegt. Der Bezug zwischen Objekten wird durch einen Link auf die Datei abgebildet. Somit fehlt in der Ursprungsdatei die Information über das Objekt in der Zielfile - es existiert typischerweise nur die Information, dass dort ein oder mehrere Objekte abgelegt sind.

Um die Hantierung von großen XML-Datenmengen, die Objekte mit ihren Daten beinhalten, zu optimieren, kann die Ablage dieser Daten auf mehrere Dateien aufgeteilt werden. Hierfür wird ein XML-Schema zur Ablage von Objekten und ihrer Bestandteile definiert. Oberhalb der Ebene des reinen XML wird somit eine weitere, objektbasierte logische Ebene eingeführt. Auf dieser Ebene ist es möglich, Objekte bzw. Teile von Objekten auf mehrere Dateien zu verteilen. Dabei müssen nicht mehr alle Objekte in einer Gesamtdatei abgelegt werden. Stattdessen ist es möglich, Objekte so abzulegen, dass die Kerninformation, die notwendig ist, um das Objekt und dessen Typ zu identifizieren, in einem Ursprungsfile vorhanden ist. Die eigentliche (meist umfangreiche) Nutzinformation des Objekts wird jedoch in ein Auslagerungsfile ausgelagert. In diesem können Daten von einem oder mehreren Objekten abgelegt sein. Hierbei macht man sich zu Nutze, dass Objekte meist aus verschiedenen „Arten von Daten“ bestehen. So kann man diese unterscheiden in Daten,

- die das Objekt an sich beschreiben (Objektidentifikation, Name, etc.),
- die von allgemeinem Interesse sind und damit für verschiedene Applikationen bzw. Teile von Applikationen von Interesse sind und
- die sehr spezifisch sind und nur für eine bestimmte Applikation bzw. eine Teil-Applikation von Interesse sind.

Dies kann man ausnutzen, um ein Objekt in logische Bestandteile aufzuteilen und diese ggf. entsprechend den Hauptverwendungen optimiert in verschiedenen Dateien abzulegen. Tools, die bestimmte Teile des Objektgeflechts nicht benötigen, können so die entsprechenden Stellen sehr leicht überlesen, da die Informationen in separate Auslagerungsfiles abgelegt werden. Diese Teile müssen vom XML-Parser nicht eingelesen und bearbeitet werden. Die Daten eines Objekts werden dementsprechend in verschiedene Bestandteile aufgeteilt, die logische Einheiten bilden und bestimmte Aspekte eines Objekts repräsentieren. Grundlage der

Gruppierung ist die logische Zusammengehörigkeit der Bestandteile des Objekts zu einer bestimmten „Sicht“ (z. B. HMI, Hardware, Software) auf das Objekt. Diese Bestandteile werden im Folgenden auch als Features bezeichnet. Sie

5 gruppieren die Parameter, Referenzen, etc. des Objekts. Über der syntaktischen Ebene von reinem XML wird also ein logisches Objektmodell und ein Mechanismus zur Aufteilung von Objektdaten definiert, der es gestattet Objektgeflechte in

10 hierarchisch strukturierte Dateien abzulegen und dabei die Daten von Objekten auf mehrere Dateien zu verteilen, die den unterschiedlichen Anforderungen für den Datenzugriff genügen, d. h. die wichtigsten UseCases für die Nutzung der Daten unterstützen.

Dem liegen folgenden Grundideen zugrunde:

- 15 - Die in XML abzulegenden Daten werden als Objekte modelliert und können über XML-Schema beschrieben werden. Hierdurch ist es möglich eine Semantik für die Auslagerung von Objekten zu definieren. Dabei ist es sinnvoll, dass
- 20 alle Objekttypen im XML-Schema von einem Basis-Objekttyp abgeleitet werden. Dies ist jedoch nicht zwingend erforderlich.
- Es wird ein Mechanismus festgelegt, wie Objekte bzw. Objektgeflechte auf mehrere Dateien aufgeteilt werden können.
- 25 - Die Auftrennung zwischen Dateien erfolgt an solchen Stellen, an denen im Objektmodell eine PartOf-Beziehung herrscht. Die Annahme hierbei ist, dass, wenn ein Objekt aus weiteren Subobjekten besteht, diese Subobjekte typischerweise Kandidaten für die Auslagerung darstellen.
- 30 Applikationen bzw. Teile von Applikationen greifen häufig auf unterschiedlicher Granularitätsstufe auf Objekte zu. So interessiert in der einen Applikation beispielsweise nur um welches Objekt es sich handelt. Erst zur
- 35 Bearbeitung des Objekts (z. B. in einem Editor) sind die Subobjekte dieses Objekts von Interesse. Erst dann wird auf diese Teildaten zugegriffen.

- In der Ursprungsdatei verbleibt ein Objektrumpf in Form einer Auslagerungsreferenz. Dies stellt sicher, dass Referenzen auf das ausgelagerte Objekt sich nicht von anderen Objektreferenzen unterscheiden. Es muss am
5 Ursprung der Referenz nicht bekannt sein, dass es sich bei dem Zielobjekt um ein ausgelagertes Objekt handelt. In der Auslagerungsdatei wird das ausgelagerte Objekt als Ganzes abgelegt. Das hat folgende Vorteile:
 - Referenzen auf ausgelagerte Objekte müssen sich nicht
10 unterscheiden von Referenzen auf nicht ausgelagerte Objekte.
 - Objekt kann verschoben werden, ohne dass Referenzen auf das Objekt zu ändern sind.
 - Navigierbarkeit auf das Objekt innerhalb der
15 Ursprungsdatei bzw. von Außen ist gegeben
- Es können auch Teile eines Objekts (Features) in eine Datei ausgelagert werden. In der Ursprungsdatei verbleibt in diesem Fall das Objekt. Nur ein oder mehrere Features des Objekts werden ausgelagert. In der Auslagerungsdatei
20 werden die Features in einem ObjektSurrogate abgelegt. Dieses Stellvertreterobjekt repräsentiert in generischer Weise eine Hülle für die Daten des Objekts und bildet den Kontext für die Ablage der Features. Der Kontext ist dabei die Identifikation des Objekts, wie es in der
25 Ursprungsdatei identifiziert wird. Das hat folgende Vorteile:
 - Es gibt einen Stellvertreter in der Auslagerungsdatei, der beschreibt zu welchem Objekt die Daten gehören.
 - Der Stellvertreter ist Objekttyp, der ein generisches
30 Objekt repräsentiert und beliebige Features aufnehmen kann.
 - Navigierbarkeit auf das Objekt innerhalb der Ursprungsdatei bis hin zum ausgelagerten Objektteil (Feature) ist gegeben.
 - Verschiebbarkeit des Objektteils ohne Referenzen hierauf
35 ändern zu müssen (Objekt enthält Rumpfinformationen über ausgelagertes Objektteil).

- In der Auslagerungsdatei stehen die Daten des Objekts nicht alleine, sondern haben durch das Stellvertreter-Objekt einen Bezug zum eigentlichen Objekt in der Ursprungsdatei (Eine Art Rückwärtsreferenz)

5

Die Referenzen auf ausgelagerte Objekte beinhalten verschiedene Daten (als XML-Attribute oder ggf. auch als XML-Elemente):

- Identifikationsdaten des Objekts (z. B. Objekt-ID, Objekt-Name, etc.)
- Zieldatei in der das Objekt zu finden ist (z. B. Name und Pfad der Datei)
- Identifikationsdaten des Objekts in der Zieldatei (z. B. Objekt-ID, Objekt-Name)

15

Durch diesen Aufbau der Referenz kann die Adressierung des Objekts in der Auslagerungsdatei unabhängig von der Identifikation des Objekts geändert werden. Regeln, an welchen Stellen in XML-Dateien abgelegte Objekte bzw. Objektgeflechte aufzuteilen sind, sind spezifisch für den Anwendungsfall zu definieren und in ein entsprechendes XML-Schema umzusetzen.

20

Ein Beispiel für die Anwendung der Erfindung ist der Export von Daten aus einer Applikation, um diese in anderen Applikationen weiterverarbeiten zu können. In einer Applikation werden Objekte in Bäumen strukturiert (mit Querverweisen durch Referenzen auf beliebige Objekte). Eine Auslagerung von Objekten in andere Dateien erfolgt nur an Teilbaumgrenzen. Alle Objekte und Features auf oberster logischer Ebene in einer ausgelagerten Datei gehören deshalb zum gleichen Objekt/Feature in der ursprünglichen Datei. Dadurch entsteht bei der Aufteilung des XML-Exports in mehrere Dateien automatisch eine baumartige Hierarchie von Dateien. Es gibt mehrere Möglichkeiten, wie man ein (logisch zusammengehörendes) Objektgeflecht auf mehrere Dateien verteilen kann:

30

35

- Aufteilung objektgranular: SubObjekte, die zu einem Objekt gehören werden nicht in der ursprünglichen Datei eingebettet, sondern in eine externe Datei geschrieben.
- Aufteilung an Feature-Grenzen: Einzelne Features werden in getrennten Dateien abgelegt. Dies ist z. B. dann vorteilhaft, wenn eine Anwendung ihre Daten in eigene Features ablegt, die im Wesentlichen nur für diese Applikation relevant sind, nicht aber für andere. Falls diese in einer eigenen Datei liegen, braucht die Anwendung nur diese Datei zu lesen.

Eine Datei, die ausgelagerte Objekte enthält, unterscheidet sich in ihrem Aufbau nicht von anderen Dateien, in denen Objekte abgelegt werden. Jede XML-Datei beginnt mit einem Standard-Header zur Identifikation, dass diese XML-Datei zu einer Menge von Export-Dateien gehört, die das abgelegte Objektgeflecht beinhalten.

Zusätzliche Vorteile bietet es, die hierarchischen Abhängigkeiten zwischen den Dateien in einem Export eines Objektgeflechts explizit mit anzugeben. Dies ist jedoch nicht notwendig und bietet nur zusätzlichen Nutzen dadurch, dass eine beliebige Datei des Exports als Einstieg für die Bearbeitung verwendet werden kann. Es bietet eine einfache Navigation auf Dateiebene, um von jeder beliebigen Datei aus zum Wurzelelement bzw. zum direkten (logischen) Parent (= Elternelement) der Datei zu gelangen. Hierzu wird für den Aufbau einer (Export-)Datendatei (z. B. <Document>) ein Standard-Header definiert. In diesem Header können zwei optionale Attribute Parent und Root vorgesehen werden. Mit Parent gibt man die nächsthöhere Datei der Hierarchie an, mit Root direkt die Wurzel, also das oberste Element der Hierarchie. Wenn man diese beiden Attribute verwendet, ist es möglich, von einer beliebigen Datei innerhalb eines Exports zur Wurzel des Exports bzw. zu dem File zu gelangen, von dem die Objektdaten der aktuellen Datei referenziert werden.

Der Aufbau des Headers und der gesamten XML Datei kann dabei über XML-Schema festgelegt werden. Ein Beispiel für eine Instanz einer Exportdatei kann wie folgt aussehen (siehe auch FIG 1).

5

Beispiel-Datei Racks.xml 20:

```
<Document
```

```
  xmlns:base="http://www.siemens.com/Industry/2001/Automation/Base"
```

```
    ...
```

10

```
      Parent="HWKonfigExport.xml" Root="HWKonfigExport.xml">
```

```
    <FileInfo Version="1.2">
```

```
      ...
```

```
    </FileInfo>
```

```
</Document>
```

15

Die Datei Racks.xml 20 ist Teil eines XML-Exports, dessen Wurzel die Datei HWKonfigExport.xml 10 bildet. Diese Datei 10 ist gleichzeitig der Vaterknoten von Racks.xml 20 im Baum des XML-Exports. Die Parent-Beziehung und die Root-Beziehung sind in FIG 1 durch die mit den Bezugszeichen 2 bzw. 3 bezeichneten Pfeile dargestellt.

Lagert man ein Objekt mit seinen Daten in eine eigene Datei aus, so benötigt man an der Stelle, wo "normalerweise" das Objekt eingebettet wäre, eine spezielle Referenz 13 (ReferencePartOfT). Diese Referenz 13 gibt an, dass es sich um eine Auslagerung 23 handelt. Die Referenz 13 spezifiziert dabei welches Objekt ausgelagert wurde und in welcher Datei es zu finden ist. Die Beziehung zwischen Referenz 13 auf der einen Seite und Auslagerung 23 auf der anderen Seite ist in FIG 1 durch einen Pfeil mit dem Bezugszeichen 1 dargestellt. Ein Beispiel für eine Schema-Definition einer Auslagerungsreferenz könnte wie folgt aussehen:

35

```
<xsd:complexType name="ReferencePartOfT">
```

```
  <xsd:complexContent>
```

```

    <xsd:attribute name="Name"          type="xsd:string" use="optional"/>
    <xsd:attribute name="Target"        type="xsd:string" use="required"/>
    <xsd:attribute name="TargetID"      type="IdT"          use="required"/>
    <xsd:attribute name="TargetName"    type="xsd:string" use="required"/>
5      </xsd:complexContent>
</xsd:complexType>

```

Die Attribute TargetID 11 und TargetName 12 enthalten die ID 21 und den Namen 22 des ausgelagerten Objekts, auf das die Referenz 13 verweist. Die ID 21 wird für die Bildung von absoluten Referenzen von einer anderen Stelle auf das Objekt benötigt. Dem Objekt kann auch ein Name 22 gegeben werden, den man ebenfalls zur Referenzierung verwenden kann. Auch wenn ein Objekt ausgelagert wird, sind durch diese beiden Attribute TargetName 12 und TargetID 11 der Name 22 bzw. die ID 21 noch im Hauptdokument vorhanden. Dies hat den Vorteil, das alle Referenzierungen auf dieses Objekt in dem File navigiert werden können. D. h. wenn das Ursprungsfile des Objekts von einer Applikation eingelesen/verarbeitet wird, so können Referenzen auf das Objekt aufgelöst und das Objekt bei Bedarf aus der ausgelagerten Datei gelesen werden.

Der Einsatz von Auslagerungsreferenzen ist sehr einfach, es wird nur das eingebettete Objekt (im Beispiel das "Rack") durch eine Auslagerungsreferenz (im Beispiel "RackLink") ersetzt. Das Element wird im produktspezifischen Schema als Element vom Typ product:ReferencePartOfT definiert.

Beispiel für den Einsatz der Auslagerungsreferenz:

```

30 <base:Station ID="1234" Name="S7300">
    <base:StructuralFeature>
        <base:RackLink TargetName="UR"
            TargetID="4711"
            Target="../Drehen/Racks.xml#4711"/>
35 </base:StructuralFeature>
</base:Station>

```

Die Definition der Auslagerung des Objekts (Rack) kann im XML-Schema für das Ursprungsobjekt in diesem Fall wie folgt definiert werden:

```
<xsd:element name="RackLink" type="ReferencePartOfT" />
```

5

Die Auslagerung von einzelnen Features eines Objekts führt dazu, dass in der Auslagerungsdatei eigentlich kein vollständiges Objekt mehr vorhanden ist, sondern nur ein Teil des Objekts. An der Stelle im Ursprungsdocument, an der sonst das Feature abgelegt würde, steht nur noch ein Link auf das ausgelagerte Feature. Beispiel:

10

```
<SubSystem ID="100" Name="DP-Master">
  <DisplayNameFeature>
    ...
  <DisplayNameFeature>
    <ProfibusFeatureLink
      Target="Feature.xml#100/feature(ProfibusFeature)"/>
    ...
</SubSystem>
```

15

20

In der Auslagerungsdatei steht das eigentliche Feature. Damit dieses Feature wieder einem Objekt zugeordnet werden kann, wird dieses Feature in einer Standard-Objekthülle (auch ObjectSurrogate genannt) abgelegt. Beispiel für das in obigem Beispiel ausgelagerte ProfibusFeature:

```
<ObjectSurrogate ID="100" Name="DP-Master">
  <ProfibusFeature>
    <GROUP_IDENT_SUM_ALL_SLAVES Value="255"/>
    <GROUP_SYNC_PROP Value="255"/>
    <GROUP_FREEZE_PROP Value="255"/>
    <LAST_USED_PROFIBUS_ADDR Value="12"/>
    <Address Value="12"/>
  </ProfibusFeature>
</ObjectSurrogate>
```

30

35

Diese Objekthülle (ObjectSurrogate) ist ein generischer Container zur Aufnahme von ausgelagerten Features. Er ist nicht spezifisch für den Applikations-Objekttyp, dessen Daten er enthält. Die Objekthülle dient zur Etablierung des

5 entsprechenden Kontexts für den Objektteil und beinhaltet die Identifikation des Objekts (ID und Name). Wie man in den Instanzen sieht, sind ID und Name in der Haupt-Datei und in der Auslagerungsdatei jeweils gleich. FIG 2 verdeutlicht diesen Zusammenhang. Mit dem Bezugszeichen 50 wird die

10 ursprüngliche Situation gekennzeichnet, wenn alles in einer Datei abgelegt wird: Das Objekt SubSystem 51 hat die beiden Features DisplayNameFeature 52 und ProfibusFeature 53. Mit dem Bezugszeichen 60 wird die Konstellation bezeichnet, bei der das ProfibusFeature 63 in eine eigene Datei 65

15 ausgelagert ist. Die Definitionen der benötigten Typen im XML-Schema könnte beispielhaft folgendermaßen aussehen:

```

<xsd:complexType name="ObjectSurrogateT ">
  <xsd:annotation>
    <xsd:documentation>object that contains features in partial
20 exports</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:restriction base="ObjectT">
      <xsd:sequence>
        <xsd:element name="App_Id" type="ApplicationSpecificIdT" minOccurs="0"
5 maxOccurs="unbounded"/>
        ...
        <xsd:element ref="Feature" maxOccurs="unbounded"/>
30 </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="Feature" type="FeatureT"/>
35 <xsd:element name="ProfibusFeature" type="ProfibusFeatureT"
  substitutionGroup="Feature">
    <xsd:attribute name="Name" type="xsd:QName" use="optional"/>

```

```
<xsd:attribute name="Target" type="xsd:string" use="required"/>  
<xsd:attribute name="Type" type="ReferenceTypeEnumT" use="optional"  
fixed="PartOf"/>  
</xsd:complexType>
```

5

Der Typ von ProfibusFeature ist vom Grundtyp FeatureT
abgeleitet. Da bei der Elementdeklaration die
SubstitutionGroup Feature angegeben wurde, kann das Element
in das ObjectSurrogate an Stelle von Feature eingehängt
werden.

10

Die Hantierung von Auslagerungen kann bei einer konkreten
Implementierung durch eine Supportbibliothek unterstützt
werden. Diese kann sowohl beim Lesen der XML-Daten, als auch
beim Schreiben automatisch die Aufteilung der XML-Daten auf
Files hantieren und den Mechanismus für den Anwender
verbergen. Aus seiner Sicht operiert er ausschließlich auf
dem Objektmodell. Verwaltung von Dateien und Referenzen
erfolgt durch die Supportbibliothek. Voraussetzung hierfür
ist, dass entsprechende Schemas für die Applikationsdaten
existieren und diese von der Supportbibliothek benutzt
werden.

15

20

Zusammenfassend betrifft die Erfindung somit ein System sowie
ein Verfahren zur vereinfachten Verwaltung von mit einer
erweiterbaren Auszeichnungssprache beschriebenen Daten. Dabei
werden die Daten in Form von Objekten strukturiert, wobei
Bestandteile der Objekte in ersten Dateien speicherbar sind,
wobei die Bestandteile jeweils eine logische Einheit eines
Objekts abbilden und wobei eine zweite Datei mit ersten
Mitteln zur Referenzierung der Bestandteile als
übergeordnete, objektbasierte logische Ebene zur Speicherung
der Objekte vorgesehen ist.

30

Patentansprüche

1. Verfahren zur Verwaltung von mit einer erweiterbaren
Auszeichnungssprache beschriebenen Daten, wobei die Daten in
5 Form von Objekten strukturiert werden, wobei Bestandteile der
Objekte in ersten Dateien speicherbar sind, wobei die
Bestandteile jeweils eine logische Einheit eines Objekts
abbilden und wobei eine zweite Datei mit ersten Mitteln zur
Referenzierung der Bestandteile als übergeordnete,
10 objektbasierte logische Ebene zur Speicherung der Objekte
vorgesehen ist.

2. Verfahren nach Anspruch 1,
dadurch gekennzeichnet,
15 dass die Bestandteile selbst Objekte sind.

3. Verfahren nach Anspruch 1,
dadurch gekennzeichnet,
dass die Bestandteile in objektspezifischen generischen
20 Containern gespeichert werden, wobei die Container zur
Referenzierung des jeweiligen Objekts dienen.

4. System zur Verwaltung von mit einer erweiterbaren
Auszeichnungssprache beschriebenen Daten, wobei Objekte zur
Strukturierung der Daten vorgesehen sind, wobei Bestandteile
der Objekte in ersten Dateien speicherbar sind, wobei die
Bestandteile jeweils eine logische Einheit eines Objekts
abbilden und wobei eine zweite Datei mit ersten Mitteln zur
Referenzierung der Bestandteile als übergeordnete,
30 objektbasierte logische Ebene zur Speicherung der Objekte
vorgesehen ist.

5. System nach Anspruch 4,
dadurch gekennzeichnet,
35 dass die Bestandteile selbst Objekte sind.

6. System nach Anspruch 4,
dadurch gekennzeichnet,
dass objektspezifische generische Container zur Speicherung
der Bestandteile der Objekte vorgesehen sind, wobei die
5 Container zur Referenzierung des jeweiligen Objekts dienen.

Zusammenfassung

Verwaltung von mit einer erweiterbaren Auszeichnungssprache
beschriebenen Daten

5

Die Erfindung betrifft ein System sowie ein Verfahren zur
vereinfachten Verwaltung von mit einer erweiterbaren
Auszeichnungssprache beschriebenen Daten. Dabei werden die
Daten in Form von Objekten strukturiert, wobei Bestandteile
10 der Objekte in ersten Dateien speicherbar sind, wobei die
Bestandteile jeweils eine logische Einheit eines Objekts
abbilden und wobei eine zweite Datei mit ersten Mitteln zur
Referenzierung der Bestandteile als übergeordnete,
objektbasierte logische Ebene zur Speicherung der Objekte
15 vorgesehen ist.

FIG 1

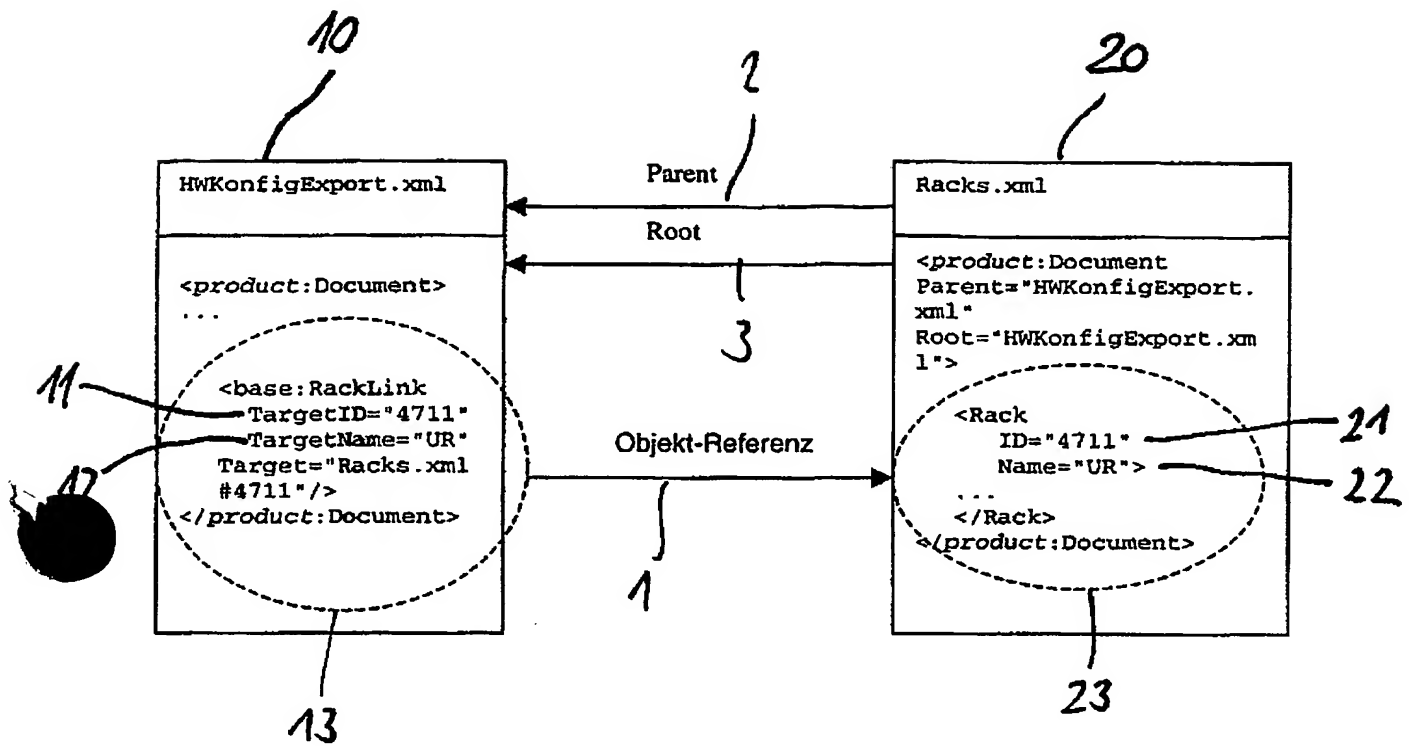


FIG 1

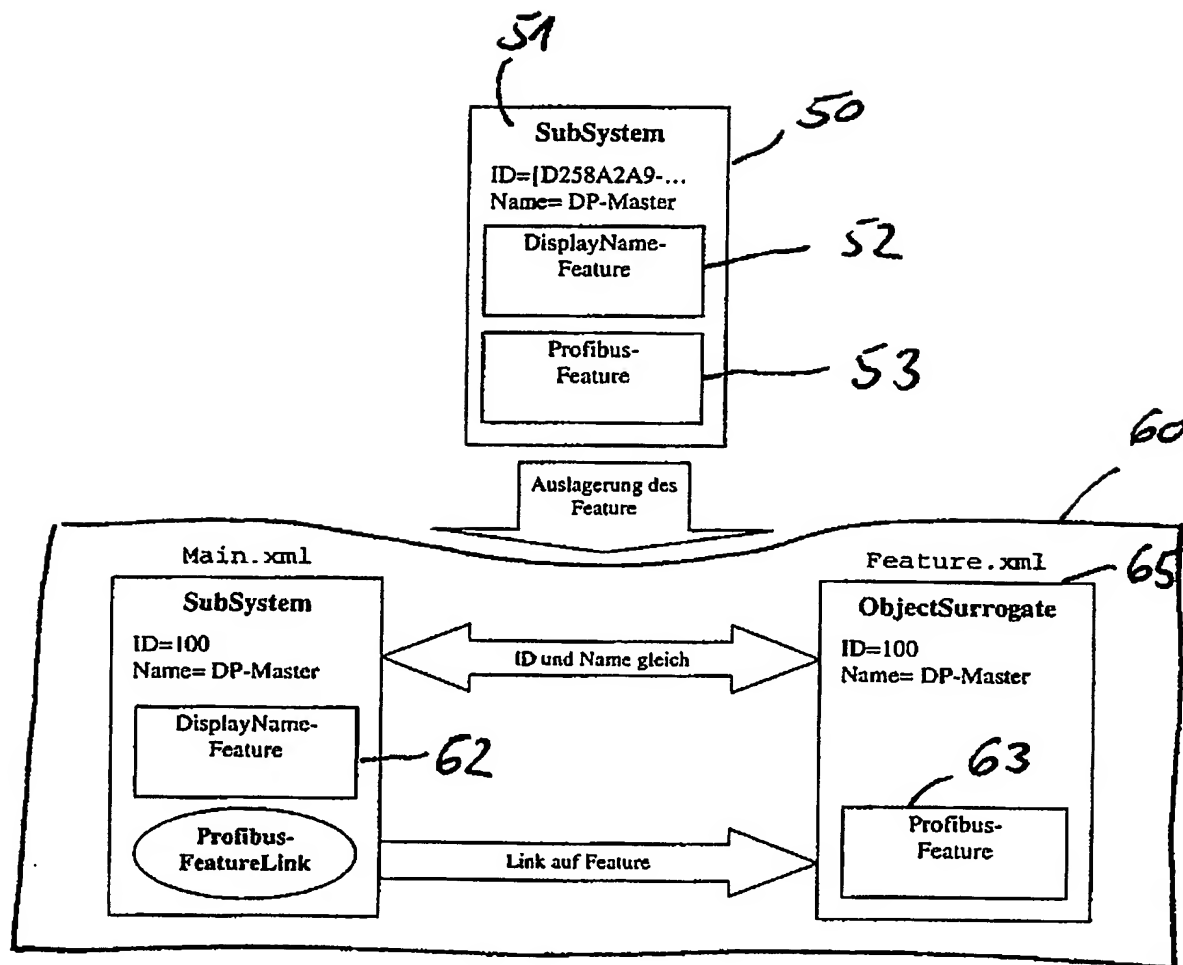


FIG 2